(2)

Technical Document 2194
October 1991

Performance Modeling of the ADA Rendezvous

A. E. Sterrett
M. K. Minei

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

This task was carried out by the Computer Software Technology Branch, Code 411, of the Naval Ocean Systems Center, under the Independent Exploratory Development (IED) program. Sponsorship was provided by the Office of Chief of Naval Research under program element 0602936N, work unit DN300189.

Released by
Gerald Schulte, Head
Computer Software
Technology Branch

Under authority of
A. G. Justice, Head
Information Processing
and Display Division

PK

CONTENTS

FIGURES

TABLES

INTRODUCTION

The Ada Programming Language (United States Department of Defense, 1983) was designed to meet the need for a standard computer-programming language. Ada has the ability to take advantage of multiprocessor environments. One feature, known as the rendezvous, allows tasks to synchronize. This very important and powerful feature is poorly understood. Rendezvous performance is a known area of concern, especially if the system is in a multiprocessor environment. In this paper, analytic techniques are developed that will predict the performance of systems using the rendezvous.

In a distributed Ada system, the rendezvous provides synchronized communication between asynchronous tasks. A system of this sort would consist of at least two processors, each serving various tasks. We have analytically developed algorithms that determine the average Rendezvous Response Time for a two-processor system. Rendezvous Response Time will be defined as the amount of time one task (i.e., a client task) must wait until its rendezvous request to another task (i.e., a server task) is completed. This approach uses Mean Value Analysis (MVA), analytic extensions to MVA, and elementary queueing theory to decompose the rendezvous into interacting separate models. The notion of a software server that maintains a queue and services rendezvous requests will play an important part in the solution techniques.

This paper is divided into three parts. First, an open model to a two-processor system is analyzed. Second, a closed model to a two-processor system is analyzed. Third, more complicated client-server rendezvous situations and possible analytic solutions are proposed. The solutions are based on the simpler client-server model of this paper. The testing of their validity will be for future work on this project.

BACKGROUND

THE ADA ENTRY CALL

Ada tasks are asynchronous processes that can execute concurrently with one another on a multiprocessor system. The feature used for task communications and synchronization is the rendezvous.

A task makes a request for rendezvous by calling an entry to another task. When the called task (i.e., the server task) is ready to perform an accept, the rendezvous begins. If the server task is not ready to accept the rendezvous request, the calling task (i.e., the client task) is blocked from executing further on its processor. At this point, the client task releases the processor on which it is running and remains in a blocked state until its rendezvous is completed.

Figure 1 is an illustration of the states of execution of a client task. A state transition exists from running (the task is executing on the processor and the rendezvous request is made) to blocked (the task is suspended from execution and waits for the server task to perform the rendezvous). When the server task accepts and completes the execution of the rendezvous, the client task becomes ready (the task is waiting to regain access to the processor).

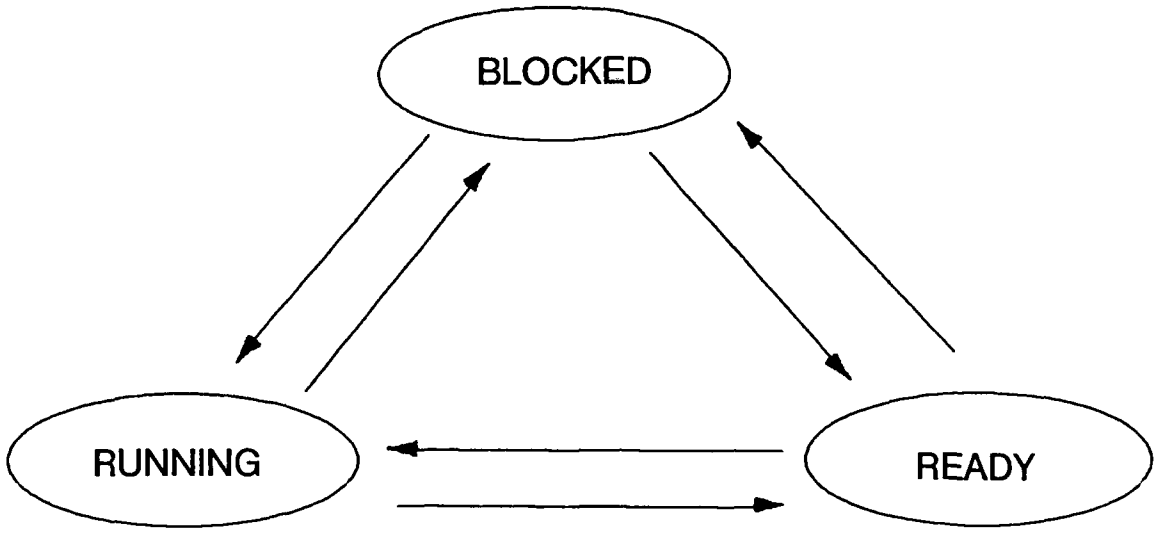

Figure 1. The states of execution of a client task on a multiprocessor system.

CHARACTERISTICS OF THE RENDEZVOUS

The rendezvous (1) represents the meeting of two tasks at the point of synchronization and (2) may communicate data. If several tasks make a call to the same entry, a queue forms and the server task will rendezvous with each client task on a first-come, first-served basis. As each rendezvous is completed, the client and server tasks continue their execution concurrently.

The following two Ada tasks running on a two-processor system are presented as our problem statement. The first task, SERVER_TASK, runs on one processor and contains only one "accept" statement. This task will make no rendezvous requests to other tasks:

```
task body SERVER_TASK is
begin
   . . .
```

```
        accept Data_Exchange (. . .) do
        . . .
        end Data_Exchange;
        . . .
    end SERVER_TASK.
```

The second task, CLIENT_TASK, runs on the other processor and contains a call to the "Data_Exchange" entry of SERVER_TASK:

```
    task body CLIENT_TASK is
    begin
        . . .
        SERVER_TASK.Data_Exchange(. . .);
        . . .
    end CLIENT_TASK.
```

CLIENT_TASK makes a call to the entry of SERVER_TASK and is blocked from further execution until SERVER_TASK completes this rendezvous. After the rendezvous is completed, CLIENT_TASK is put into a ready state. Both tasks now go their separate way and execute in parallel.

We define Rendezvous Response Time for CLIENT_TASK as follows. Let DELAY TIME be the length of time that a task spends in the rendezvous entry queue. This time begins when CLIENT_TASK makes a call to the entry of SERVER_TASK and ends when SERVER_TASK accepts this call for rendezvous. This period includes any delays due to any other tasks that have previously made the same call to entry "Data_Exchange" and are still waiting for rendezvous. Define RENDEZVOUS TIME as the length of time that begins when CLIENT_TASK is accepted for rendezvous and ends when the "end Data_Exchange;" line of SERVER_TASK is executed. Thus, define

Rendezvous Response Time = DELAY TIME + RENDEZVOUS TIME.

From figure 1, Rendezvous Response Time is equivalent to having CLIENT_TASK go from "Running" to "Blocked" to "Ready."

ANALYSIS FOR THE OPEN MODEL

An algorithm is presented for determining the average Rendezvous Response Time for the following open model to a two-processor system. Hereon, the word "average" will be assumed when appropriate.

STATEMENT OF THE PROBLEM

Consider the open model for a two-processor system as drawn in figure 2. Task SERVER_TASK will execute exclusively on one processor that will be called Server Processor. SERVER_TASK will have the following form.

```
task body SERVER_TASK is
begin
    loop
        accept Data_Exchange (. . .) do
      . . .
        end Data_Exchange;
    end loop;
end SERVER_TASK.
```

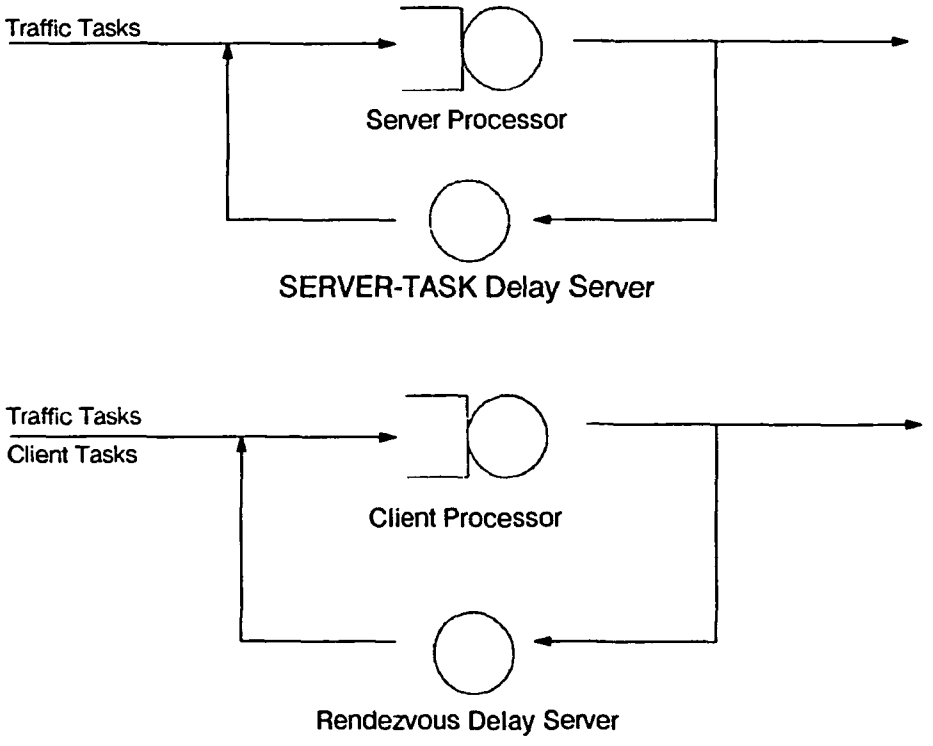

Figure 2. The open model to the two-processor system.

SERVER_TASK will execute as a reentrant task, serving one rendezvous request each time it gains access to the processor. Also, other tasks will arrive at Server Processor at some average rate and will provide contention with SERVER_TASK for processor resources. These tasks play no role in the rendezvous itself and will be called the "traffic tasks" of Server Processor. Each of these tasks will require only a finite amount of service

time before exiting the entire model. SERVER_TASK will enter and remain in the SERVER_TASK delay server whenever there are no rendezvous requests to service.

At the second processor, other tasks will arrive at some given rate. These tasks will make rendezvous requests to SERVER_TASK. Thus, we will refer to these tasks as "client tasks" and this processor as Client Processor. In this case, client tasks will have the form of CLIENT_TASK as defined below.

```
task body CLIENT_TASK is
begin
    . . .
    SERVER_TASK.Data_Exchange (. . .);
    . . .
end CLIENT_TASK.
```

A client task arrives at Client Processor and enters the processor queue. When the client task is selected for service, it makes a rendezvous request with SERVER_TASK and is put in a "blocked" state. The client task next enters the rendezvous delay server. At this point, the Client Processor is assigned to another task in the queue. Client tasks that enter into the rendezvous delay server will remain there until their rendezvous request is completed by the SERVER_TASK. Also at Client Processor, tasks that have nothing to do with the rendezvous arrive for service. These are referred to as the "traffic tasks" of Client Processor.

Assuming the forced-flow law applies here, the rate at which client tasks arrive at Client Processor is equal to the rate client tasks make rendezvous requests to SERVER_TASK. At Server Processor, SERVER_TASK rendezvous with each request one at a time on a first-come, first-served basis. When the rendezvous request is completed, the client task is released from the "blocked" state and reenters the queue at Client Processor for further execution.

There are two observations to make on the definition of the Rendezvous Response Time: (1) Rendezvous Response Time is unaffected by the processor time needed by the client tasks at Client Processor, and (2) is unaffected by the processor time needed by the "traffic tasks" arriving at Client Processor. Thus, it is not necessary to consider the service demands for the client tasks and the traffic tasks at Client Processor. We reason that analysis to Rendezvous Response Time will only require looking at the activities at Server Processor. Knowledge of the arrival rate of the rendezvous requests to SERVER_TASK, the service demands of SERVER_TASK and the traffic tasks at Server Processor, and the arrival rate of the traffic tasks to Server Processor are all that are needed.

Mean Value Analysis (MVA), analytic extensions to MVA, and elementary queueing theory were used to develop an algorithm for calculating the Rendezvous Response Time

for the the client tasks of Client Processor. Before presenting this algorithm, its input and output data will be formally introduced.

INPUT AND OUTPUT DATA FOR THE ALGORITHM

The input data are λ_H, λ_R, D_H, and D_R. They are defined as follows:

1. The traffic tasks arriving at Server Processor is a Poisson process with parameter λ_H.

2. The client tasks arriving at Client Processor is a Poisson process with parameter λ_R. Thus, the rendezvous requests made to SERVER_TASK is also a Poisson process with the same parameter.

3. The service demand at Server Processor for a traffic task is exponentially distributed with average service demand D_H.

4. The service demand at Server Processor for SERVER_TASK (i.e., RENDEZVOUS TIME between SERVER_TASK and one client task) is exponentially distributed with average service demand D_R.

Using these input values, we determined Rendezvous Response Time for the client tasks of Client Processor.

MODEL ANALYSIS AND ALGORITHM DERIVATION

An analysis of the problem using the conventional queueing approaches is difficult because it violates several device homogeneity assumptions. The assumptions assert that a customer must not be present (waiting for or consuming service) at more than one resource at the same time. Also, the ability of a resource to render service must be independent of another resource. In other words, customers and resources must be independent of each other. Neither of these assumptions are true for our problem if we view SERVER_TASK as a resource providing a rendezvous for the client tasks of Client Processor. When a client task makes its rendezvous request, the client task enters another queue, namely, the entry queue of SERVER_TASK. Thus, this client task is present at two resources. In addition, Client Processor must suspend service to the client task until the rendezvous request of the client task is completed. Thus, Client Processor is a resource dependent on SERVER_TASK.

Our approach was to use MVA to develop an algorithm that will ease these homogeneity assumptions. The idea is to view SERVER_TASK as a "shadow" or "software" server that maintains a queue of rendezvous requests and provides service to each request. SERVER_TASK, which must contend with intervening traffic, can only provide rendezvous services when it gains access to Server Processor. Thus, the service

rate of the software server will be adjusted to reflect this contention. (In Ada 9X, these servers will be used to manage rendezvous priorities.) References to shadow servers can be found in Jacobson and Lazowska (1982), Woodside et al. (1986), and Agrawal and Buzen (1983). The approach will be based on the Jacobson-Lazowska method for solving simultaneous resource possession.

The first-order approximation to the open model is as follows:

Server Processor will be looked at as a queueing center consisting of two parallel service facilities. One facility will act as the SERVER_TASK software server and process the set of rendezvous requests from the client tasks. The other facility will process the set of traffic tasks (figure 3). The residence time of the rendezvous requests class will be used to predict Rendezvous Response Time of our original open model. Keep in mind that in the original queueing network of figure 2, SERVER_TASK can complete only one rendezvous at a time. Thus, the rate that the rendezvous requests are processed at the software server of figure 3 must reflect the processing demands provided by the traffic tasks at its facility. A technique called "load concealment" was used to hide the processor demands required by the traffic tasks from the rendezvous requests .

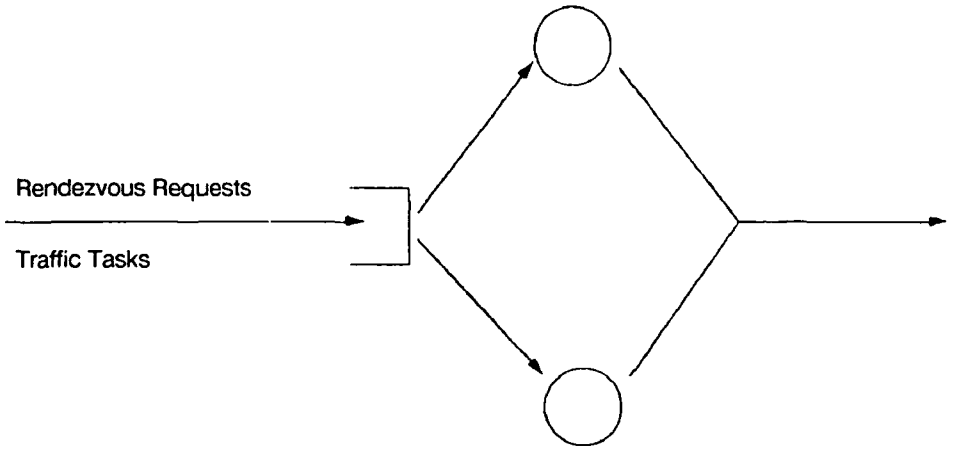

Figure 3. Server Processor separated into a queueing center consisting of two parallel service facilities.

Figure 3 can be easily solved as a simple two-stage Erlangian server (Kleinrock, 1975) with the residence time of the rendezvous requests class used to predict Rendezvous Response Time. First, the effects on the software server due to traffic tasks processor demands must be taken into account. To do this, we "inflated" the service demands of the

rendezvous requests class in figure 3. Thus, the processor utility used by the traffic tasks class are concealed from the rendezvous requests class.

The algorithm follows:

1. Define

$$\alpha_R = \lambda_R/(\lambda_R + \lambda_H)$$

$$\alpha_H = \lambda_H/(\lambda_R + \lambda_H) \ .$$

2. "Inflate" the value D_R as

$$D_R' = D_R/(1 - \lambda_H D_H) \ .$$

3. Set

$$x = \alpha_R * D_R' + \alpha_H * D_H$$

$$y = 2 * (\alpha_R * D_R'^2 + \alpha_H * D_H^2)$$

$$K = (y - x^2)/x^2 \ ,$$

and determine the total utility of the queueing center using the original service demands,

$$\rho = (\lambda_R + \lambda_H) * (\alpha_R * D_R + \alpha_H * D_H \ .$$

4. Thus, the residence time of the rendezvous requests class of figure 3 is

$$D_R = \frac{\rho x(1 + K)}{2(1 - \rho)} \ .$$

This value will be Rendezvous Response Time.

COMPUTER SIMULATION AND ALGORITHM RESULTS

The algorithm was compared for accuracy to the two different independent computer simulations of our open model. Because output data from both simulations agreed with each other, we are confident of the correctness of the simulation results.

Table 1 is a table of algorithm predictions along with a comparison to simulation runs. For these tests, $U_H = \lambda_H * D_H$ and $U_R = \lambda_R * D_R$ was defined as the rendezvous requests and traffic tasks utilizations respectively, and fixed at $D_H = D_R = 10$ and $\lambda_R = 1/50$ with varied λ_H. Thus, U_R is fixed at 20%. Figures 4 and 5 are relevant graphs from the table.

Table 1. Rendezvous Response Time results with $D_H = D_R = 10$ and $\lambda_R = 1/50$. U_R is fixed at 20% and λ_H is varied to obtain U_H.

U_H (%)	Analysis	Simulation	Relative Error (%)
5	5.26	5.18	1.52
10	5.58	5.57	0.18
20	6.44	6.30	2.17
40	10.18	8.94	12.18
60	26.55	20.07	24.41

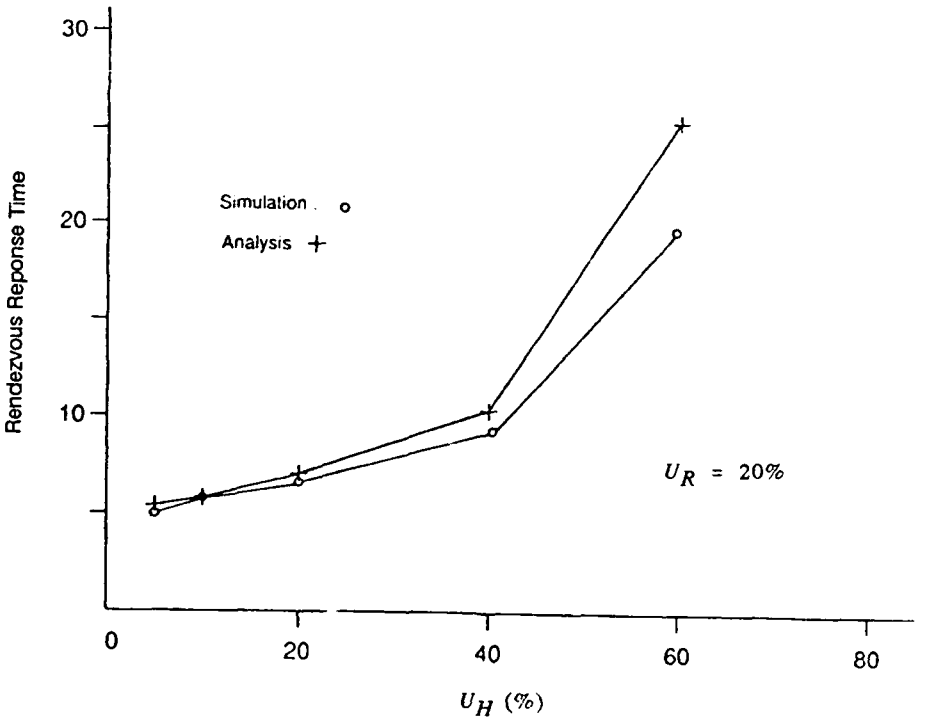

Figure 4. A graph comparison of results from table 1.

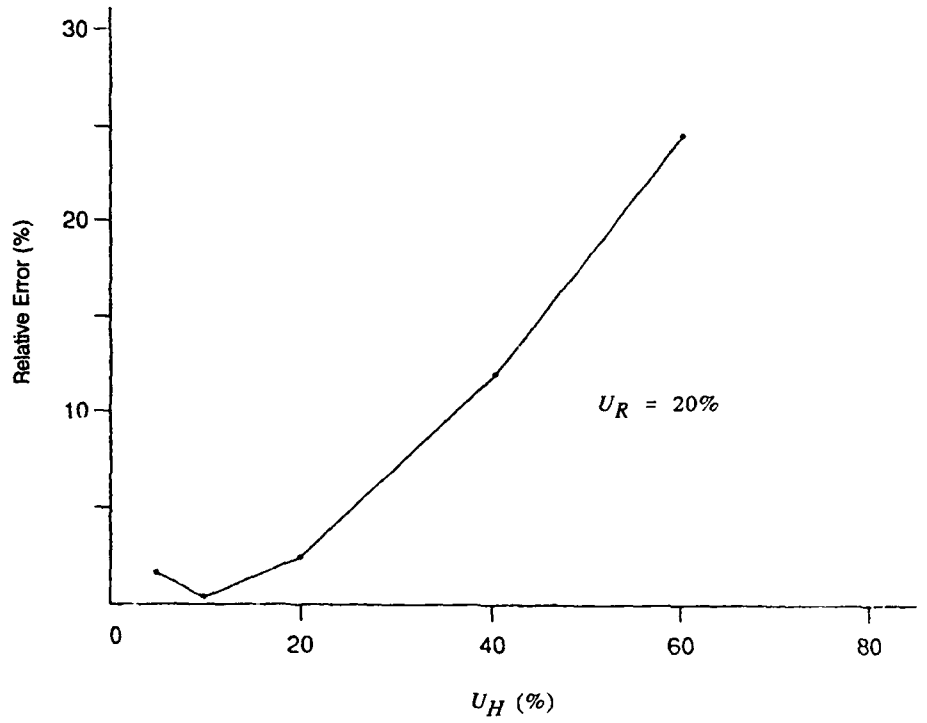

Figure 5. A graph of relative errors from table 1.

Table 2 was obtained in the same manner, but with $D_H = 2$, $D_R = 4$ and $\lambda_R = 1/20$. Figures 6 and 7 are relevant graphs from this table.

Table 2. Rendezvous Response Time results with $D_H = 2$, $D_R = 4$, and $\lambda_R = 1/50$. U_R is fixed at 20% and λ_H is varied to obtain U_H.

U_H (%)	Analysis	Simulation	Relative Error (%)
5	13.48	13.48	0
10	14.61	14.68	0.48
20	17.59	17.53	0.34
40	29.55	28.86	2.34
60	77.27	83.18	7.65

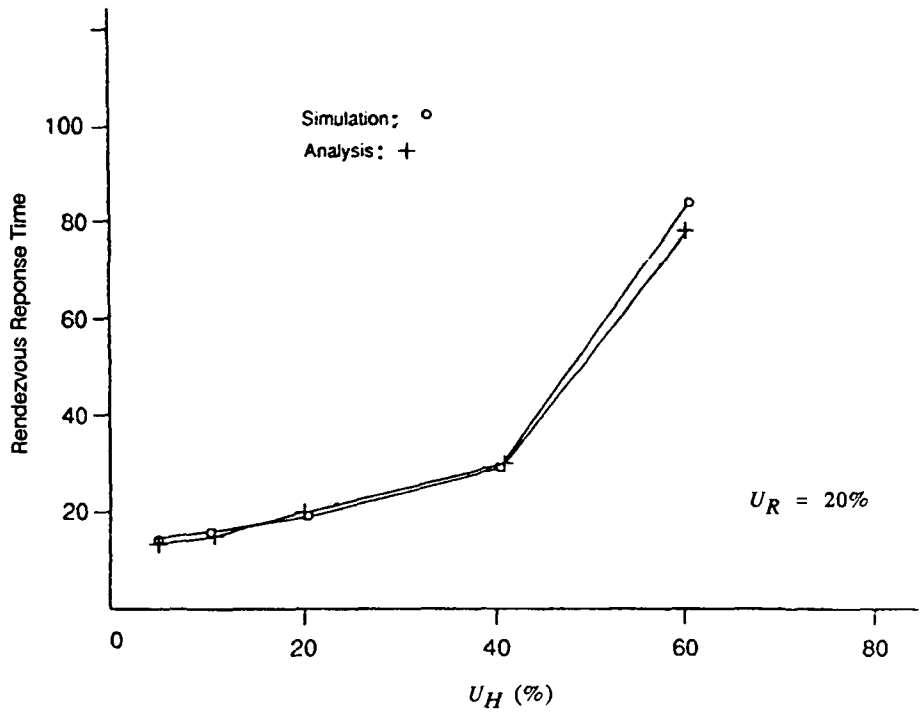

Figure 6. A graph comparison of results from table 2.

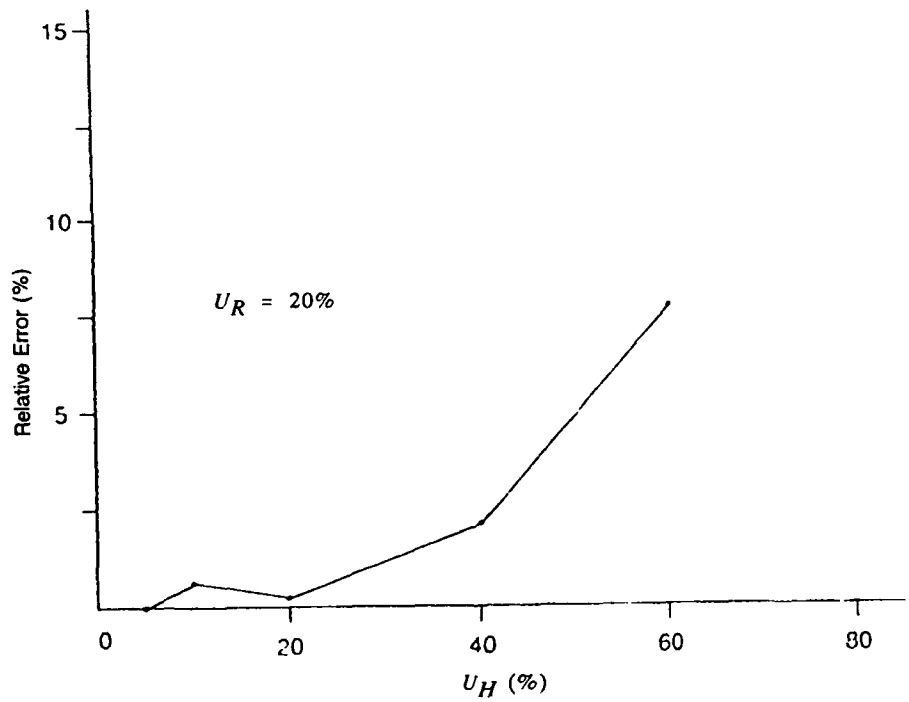

Figure 7. A graph of relative errors from table 2.

DISCUSSION OF RESULTS

Examination of the simulation data has shown that Rendezvous Response Time will increase at a slow rate until it reaches some value for U_H, at which point, an "elbowing" effect appears (figures 4 and 6). Before the elbow appears, the analysis is a good prediction to the simulated data. After the appearance of the elbow, our relative error between analysis and simulation grows. The rate that this relative error increases is a function of the input values. At this time, we are unsure of the exact nature of this dependence.

The simulation data were compared with the algorithm using no inflation techniques. These comparisons resulted in a very small relative error for data lying to the left of the elbow of the simulation. To the right of the elbow, the simulation curve increased to infinity at a faster rate than this new analytic curve. This "noninflated" algorithm actually solves figure 3 as a two-stage Erlangian server. Thus, there is a region in the total utilization of the Server Processor within which the Server Processor basically serves both its traffic tasks and the SERVER_TASK as though the traffic tasks and rendezvous requests were arriving directly at its queue for service. In this region, the effects of the rendezvous are not felt by the client tasks.

The original algorithm can be used to predict the location of the simulation curve elbow. Such information could be used for bounding the arrival rates of customers to a system. A consequence of ignoring these bounds would be an explosion in response time for systems using the rendezvous.

ANALYSIS FOR THE CLOSED MODEL

A closed model represents a system in which a task queues and receives service at the device then returns to wait in a delay server for some period of time. The time a task spends in a delay server is called "think time" because the delay server represents the amount of time a user of the system thinks before sending the task back into the system. Figure 8 is an example of a closed system.

The delay server will be used in the closed model to account for the idle time of SERVER_TASK, i.e., the time when there are no rendezvous requests arriving for SERVER_TASK.

STATEMENT OF PROBLEM

An algorithm will be developed to determine Rendezvous Response Time for the following closed model to a two-processor system. Figure 9 is our closed two-processor model.

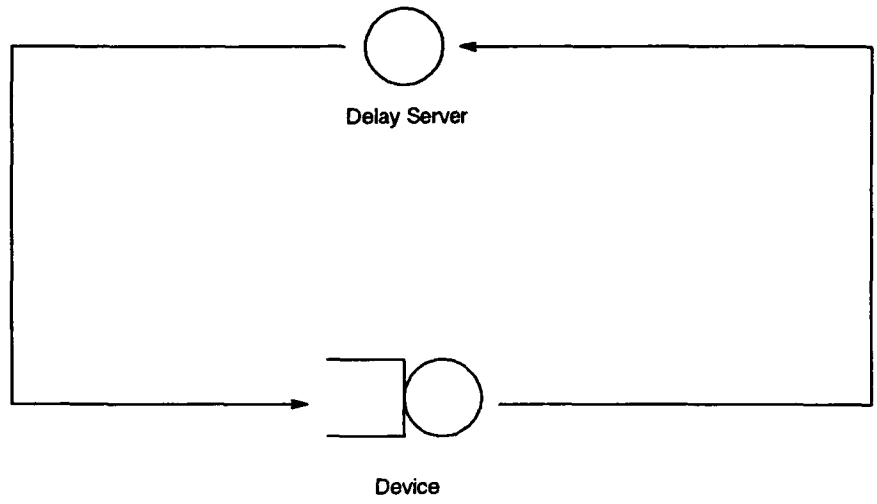

Figure 8. An example of a closed model.

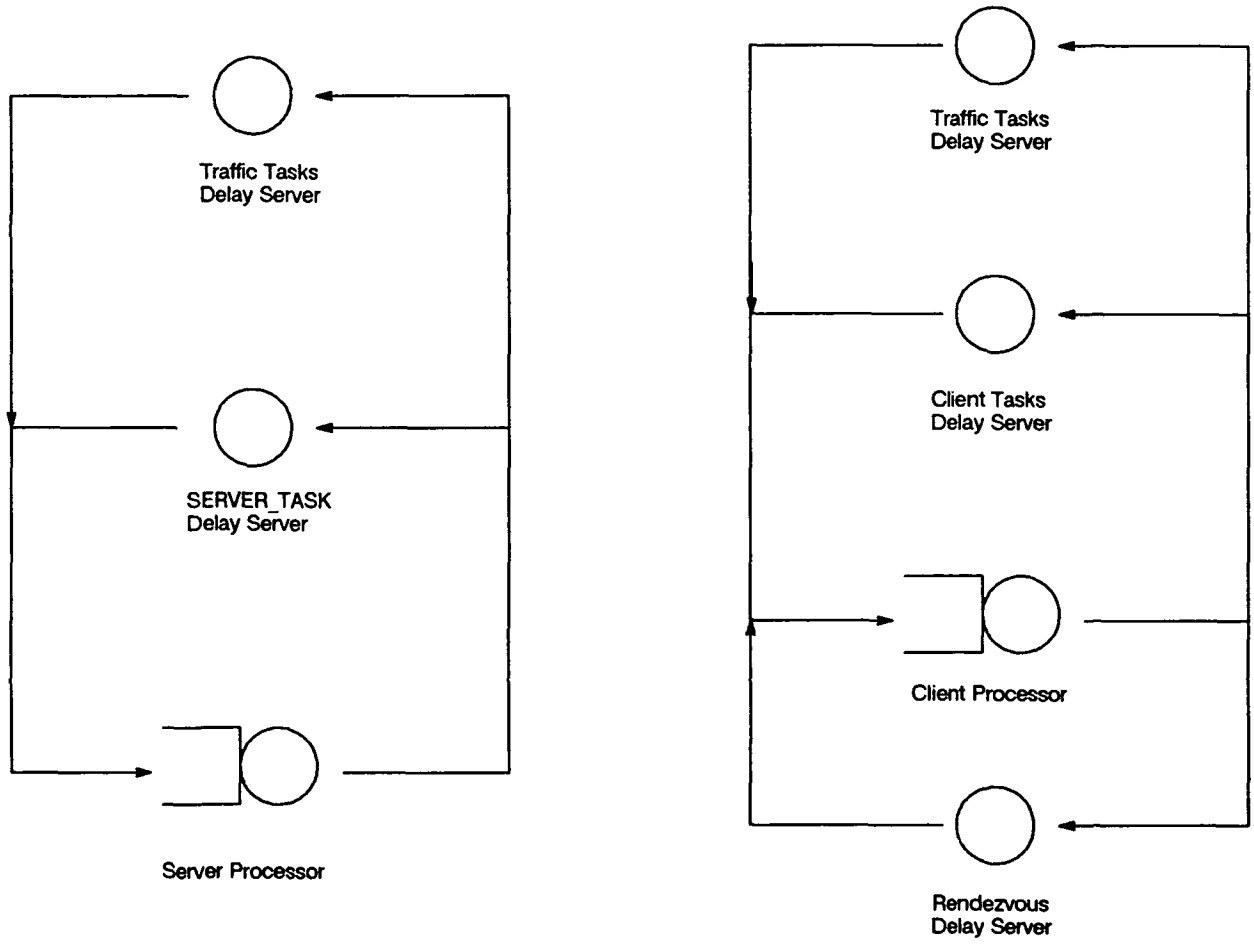

Figure 9. The closed two-processor model.

At Server Processor submodel, traffic tasks and the SERVER_TASK queue for processing time. Define $N_{Traffic\ Task,\ Server\ Processor}$ as the number of traffic tasks at Server Processor. The think time (time spent in the delay server) for each traffic task will be $Z_{Traffic\ Task,\ Server\ Processor}$. A traffic task originates from the delay server and enters the queue to Server Processor. After service at Server Processor, the traffic task will return to its delay server. SERVER_TASK is either in queue to Server Processor, executing on Server Processor, or in its delay server waiting for a client task to make a rendezvous request.

At the Client Processor submodel, both its traffic tasks and client tasks will queue for processing time. The number of traffic tasks at Client Processor is $N_{Traffic\ Task,\ Client\ Processor}$ and their think time is $Z_{Traffic\ Task,\ Client\ Processor}$. The number of client tasks is $N_{Client\ Task}$ with think time $Z_{Client\ Task}$. Once a client task makes a rendezvous request, it enters the rendezvous delay server and waits there until the request is completed. The amount of time spent in this delay server is Rendezvous Response Time.

An algorithm for determining Rendezvous Response Time for the client tasks will be developed.

INTRODUCTION OF SERVICE PHASES

In discussing closed models, it is useful to bring up the idea of service phases. The phase concept will improve the model's accuracy as it relates to an actual system. The phases of our model are defined as follows.

SERVER_TASK will have three service phases. The first phase is the service demand required by SERVER_TASK before accepting a rendezvous request. The second phase is the amount of time needed by SERVER_TASK to complete one rendezvous. This is RENDEZVOUS TIME as defined earlier. The final phase is the final service demand required by SERVER_TASK before it either requeues at Server Processor for the next rendezvous or returns to its delay server. Note that as the second phase ends and the third phase begins, the client task in rendezvous is released from its "blocked" state and is ready for further execution at its processor.

The client tasks will each have two service phases. The first phase is the service demand at Client Processor required before the client task makes its rendezvous request. The second phase is the service demand required by the client task after its rendezvous is completed but before it returns to the client-task delay server.

The traffic tasks of both Server Processor and Client Processor will have only one service phase.

INPUT AND OUTPUT DATA FOR THE CLOSED MODEL

The input data are defined as follows:

1. The number of traffic tasks at Server Processor is $N_{Traffic\ Task,\ Server\ Processor}$. The time spent in their delay server is exponentially distributed with average value $Z_{Traffic\ Task,\ Server\ Processor}$.

2. The number of traffic tasks at Client Processor is $N_{Traffic\ Task,\ Client\ Processor}$. The time spent in their delay server is exponentially distributed with average value $Z_{Traffic\ Task,\ Client\ Processor}$.

3. The number of client tasks is $N_{Client\ Task}$. The time spent in their delay server is exponentially distributed with average value $Z_{Client\ Task}$.

4. The first service phase for the SERVER_TASK is exponentially distributed with average service demand $D1_{SERVER\_TASK}$.

5. The second service phase for the SERVER_TASK is exponentially distributed with average service demand $D2_{SERVER\_TASK}$.

6. The third service phase for the SERVER_TASK is exponentially distributed with average service demand $D3_{SERVER\_TASK}$.

7. The first service phase for the client tasks is exponentially distributed with average service demand $D1_{CLIENT\_TASK}$.

8. The second service phase for the client tasks is exponentially distributed with average service demand $D2_{CLIENT\_TASK}$.

9. The service demand for the traffic tasks of Server Processor is exponentially distributed with average service demand $D_{Traffic\ Task,\ Server\ Processor}$.

10. The service demand for the traffic tasks of Client Processor is exponentially distributed with average service demand $D_{Traffic\ Task,\ Client\ Processor}$.

Using these input values, an algorithm will be developed for determining Rendezvous Response Time for the client tasks.

MODEL ANALYSIS AND ALGORITHM DERIVATION

As before, certain homogeneity assumptions are not met in our closed model. Our approach is to use MVA and extensions to develop an algorithm much like the algorithm developed for the open model. The idea is to view SERVER_TASK as a software server connected to Client Processor. The service rate of this server will be adjusted to reflect the contention that the actual SERVER_TASK must deal with at Server Processor.

Figure 10 is a conceptualization of the algorithm. In Network 1, SERVER_TASK competes with the traffic tasks of Server Processor. Each time SERVER_TASK gains access to the processor, SERVER_TASK completes all three of its service phases. After completion, SERVER_TASK will either requeue at the processor or return to its delay server and wait for the next rendezvous request to arrive.

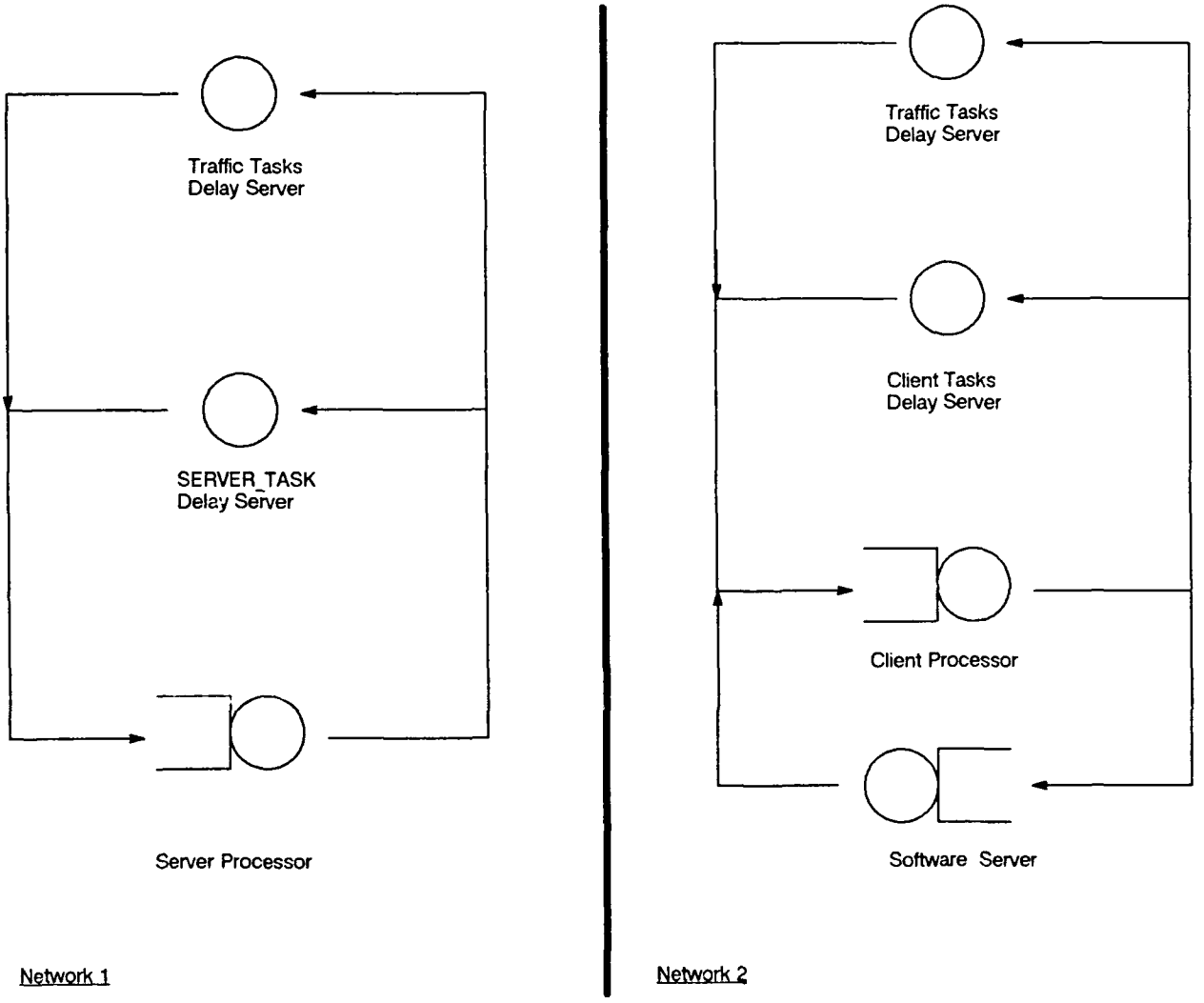

Figure 10. A conceptualization of the algorithm.

The SERVER_TASK software server of Network 2 represents SERVER_TASK found in Network 1. The residence time of a client task at this server will be Rendezvous Response Time. Each of the client tasks first visit Client Processor, then the software server then back to Client Processor before returning to the its delay server.

In the algorithm, we will iteratively solve Networks 1 and 2. For each iteration, the output of one network will become the input for the other.

1. Solve Network 2 for the following two values with the service demand at the SERVER_TASK software server equal to $D2_{SERVER\_TASK}$.

   a. $R_{Client\ Task,\ Software\ Server}$ = Residence time for the client tasks at the SERVER_TASK software server.

   b. $U_{Client\ Task,\ Software\ Server}$ = Utilization of the software server by the client tasks.

2. Using the values from the previous step, define in Network 1, the delay time of the SERVER_TASK in its delay server, Z_{SERVER_TASK} , as

$$Z_{SERVER\_TASK} = \frac{R_{Client\ Task,\ Software\ Server}}{U_{Client\ Task,\ Software\ Server}} - R_{Client\ Task,\ Software\ Server}$$

3. Solve Network 1 for $U_{Traffic\ Task,\ Server\ Processor}$ where this value is defined as the utilization of Server Processor by its traffic tasks.

4. Define the new service demand for the SERVER_TASK software server of Network 2, $D2'_{SERVER\_TASK}$, to be

$$D2'_{SERVER\_TASK} = \frac{D2_{SERVER\_TASK}}{1 - U_{Traffic\ Task,\ Server\ Processor}} .$$

5. Solve Network 2 for $R_{Client\ Task,\ Software\ Server}$ and $U_{Client\ Task,\ Software\ Server}$ and return to Step (2).

6. Iterate until $R_{Client\ Task,\ Software\ Server}$ converges to a value. This value is Rendezvous Response Time of the client tasks. From Lazowska et al. (1984), exact MVA will be used to complete steps 1, 3, and 5.

COMPUTER SIMULATION AND ALGORITHM RESULTS

Table 3 is a table comparing algorithm predictions to computer simulation results using the following input data.

1. $Z_{Traffic\ Task,\ Server\ Processor} = 50.0$
2. $Z_{Traffic\ Task,\ Client\ Processor} = 50.0$
3. $Z_{Client\ Task} = 50.0$
4. $D_{Traffic\ Task,\ Server\ Processor} = 1.0$
5. $D_{Traffic\ Task,\ Client\ Processor} = 1.0$

6. $D1_{SERVER\_TASK} = 0.5$

7. $D2_{SERVER\_TASK} = 0.5$

8. $D3_{SERVER\_TASK} = 0.0$

9. $D1_{CLIENT\_TASK} = 1.0$

10. $D2_{CLIENT\_TASK} = 0.0$

11. $N = N_{TRAFFIC\ TASK,\ SERVER\ PROCESSOR} = N_{TRAFFIC\ TASK,\ CLIENT\ PROCESSOR} = N_{TRAFFIC\ TASK,\ CLIENT\ PROCESSOR} = N_{CLIENT\ TASK}$

Table 3. Results with input values as defined above.

N	$U_{Server\_Processor}$ (%)	$U_{Client\_Processor}$ (%)	Analysis	Simulation	Relative Error (%)
5	19	19	1.231	1.21	0.00
10	35	38	1.56	1.52	3.85
15	55	56	2.17	2.33	7.37
20	62	73	3.39	4.16	22.71
25	88	85	6.37	8.13	27.63
30	86	93	14.98	17.07	13.95

The values $U_{Server\ Processor}$ and $U_{Client\ Processor}$ are analytically obtained total utilities at the processors. Figure 11 is a graph comparisons of the results and figure 12 is a graph of the relative errors.

Table 4 shows the results for the algorithm by using the same input data except with $D1_{SERVER\_TASK}$ and $D2_{SERVER\_TASK}$ set equal to 1 and $D1_{CLIENT\_TASK}$ set equal to 2.

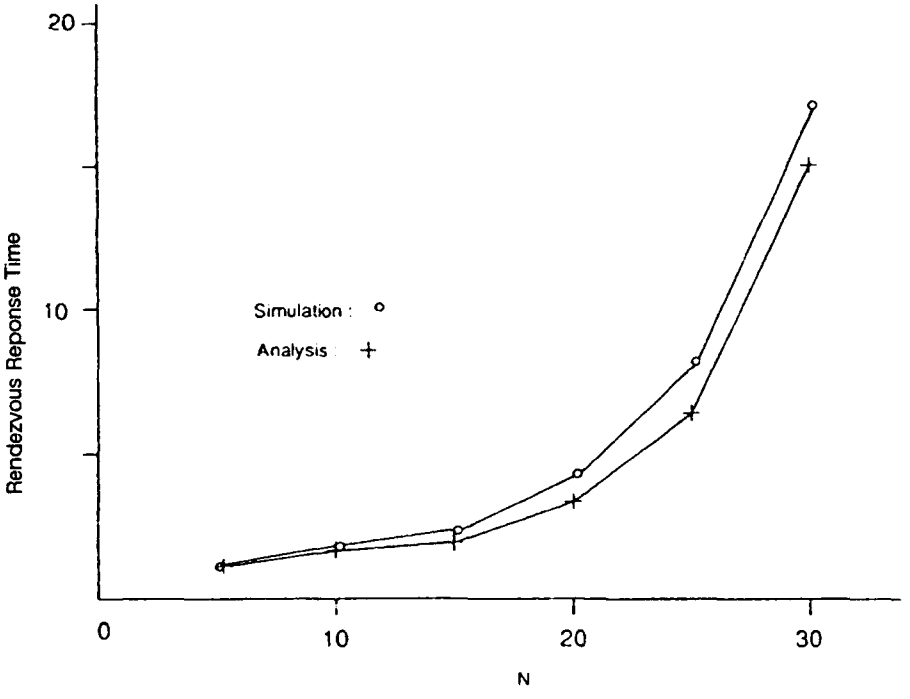

Figure 11. A graph comparison of results from table 3.

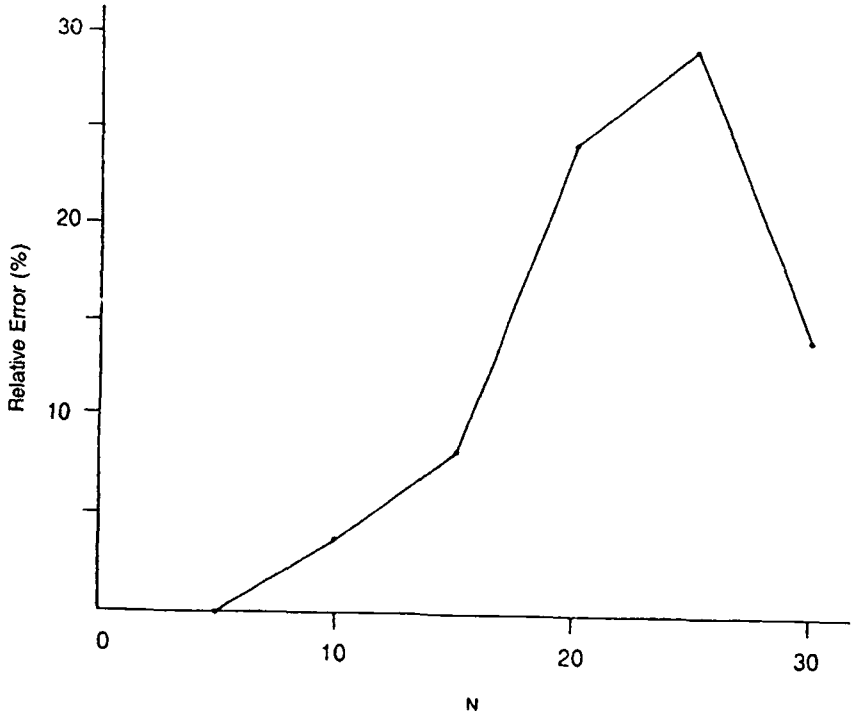

Figure 12. A graph of relative errors from table 3.

Table 4. Results with input values as defined above.

N	$U_{Server\_Processor}$ (%)	$U_{Client\_Processor}$ (%)	Analysis	Simulation	Relative Error (%)
4	21	23	2.45	2.42	1.22
8	38	43	3.29	3.12	5.17
12	53	63	4.79	4.54	5.22
16	66	78	7.79	7.60	2.44
20	79	88	14.13	14.37	1.70
24	94	92	26.98	26.94	0.15

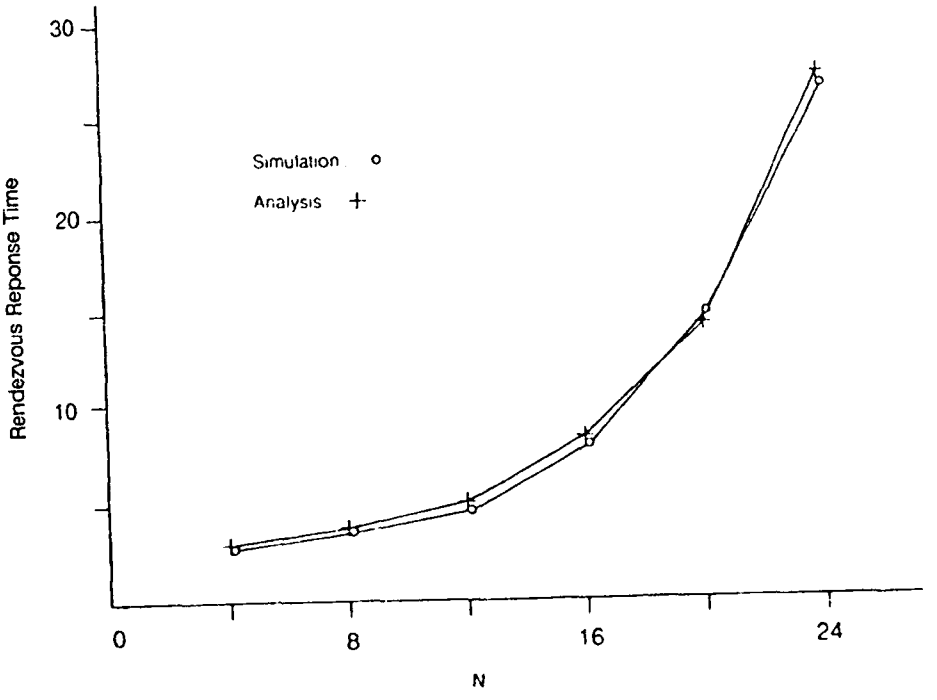

Figure 13. A graph comparison of results from table 4.

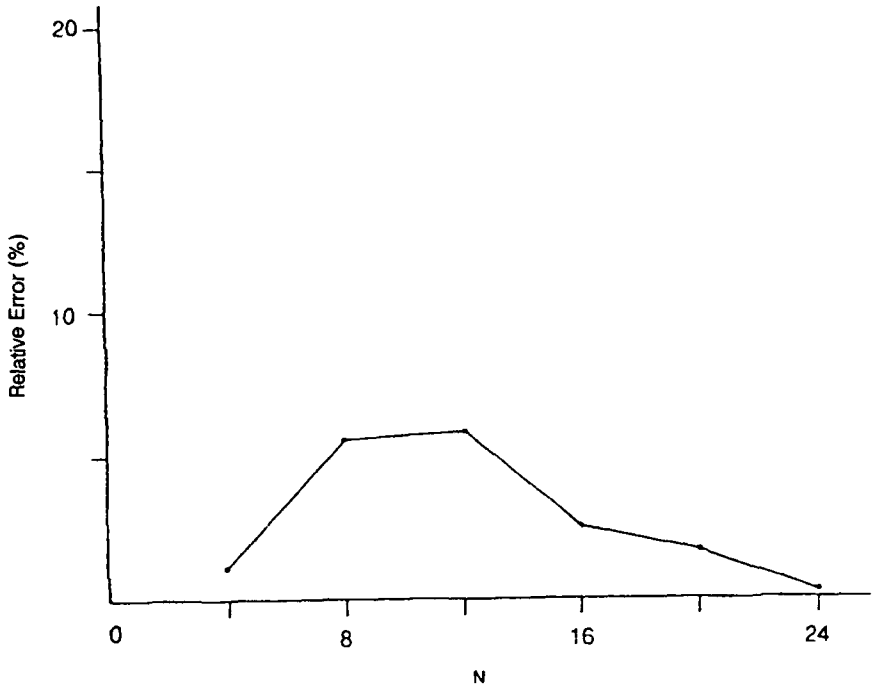

Figure 14. A graph of relative errors from table 4.

DISCUSSION OF RESULTS

An examination of the results showed that relative error remained low in most cases even if the utilizations at Server and Client Processors were near 100%. The simulated data appear to increase at a steady pace as utilities are increased.

The algorithm to the closed model is much more computationally intense and requires more input data than the algorithm for the open model. In the case of the closed model, a computer would be needed to obtain analytic values. However, the relative errors for the closed model appear to remain small even at very high-processor utilities.

In steps 1, 3, and 5, exact MVA was used to solve the closed network. This method did not take into account the fact that both the Server and Client Processors act like two-stage parallel Erlangian servers. Thus, exact MVA will only approximate the solution to the network. An error term will be incorporated into the predicted Rendezvous Response Time value for each time we iterate through the algorithm. In the future, other methods for completing steps 1, 3, and 5 will be used to determine if this improves Rendezvous Response Time predictions.

FUTURE WORK

In this section, several different client-server rendezvous situations are presented. Proposed solutions will be based on the work presented above. Testing of their validity will be done as future work for this project. For simplicity, only the open model case will be considered.

TWO RENDEZVOUS SERVERS AT SERVER PROCESSOR

Suppose two different server tasks run on Server Processor, say SERVER_TASK_1 and SERVER_TASK_2. We let client tasks (Type 1) and client tasks (Type 2) request rendezvous from SERVER_TASK_1 and SERVER_TASK_2, respectively. Define the following notation.

1. $\lambda_{R(Type\ 1)}$ = Arrival Rate for Client Tasks (Type 1) to Client Processor.
2. $\lambda_{R(Type\ 2)}$ = Arrival Rate for Client Tasks (Type 2) to Client Processor.
3. λ_H = Arrival Rate for Traffic Tasks to Server Processor.
4. $D_{R(Type\ 1)}$ = Service Demand for SERVER_TASK_1 (i.e., the RENDEZVOUS TIME Between SERVER_TASK_1 and Client Task (Type 1)).
5. $D_{R(Type\ 2)}$ = Service Demand for SERVER_TASK_2 (i.e., the RENDEZVOUS TIME Between SERVER_TASK_1 and Client Task (Type 2)).
6. D_H = Service Demand for Traffic Tasks of Server Processor.

Server Processor is separated into three parallel service facilities. Facility 1 will process the rendezvous requests of Client Tasks (Type 1), Facility 2 will process the rendezvous requests of Client Tasks (Type 2), and Facility 3 will process the traffic tasks (figure 15).

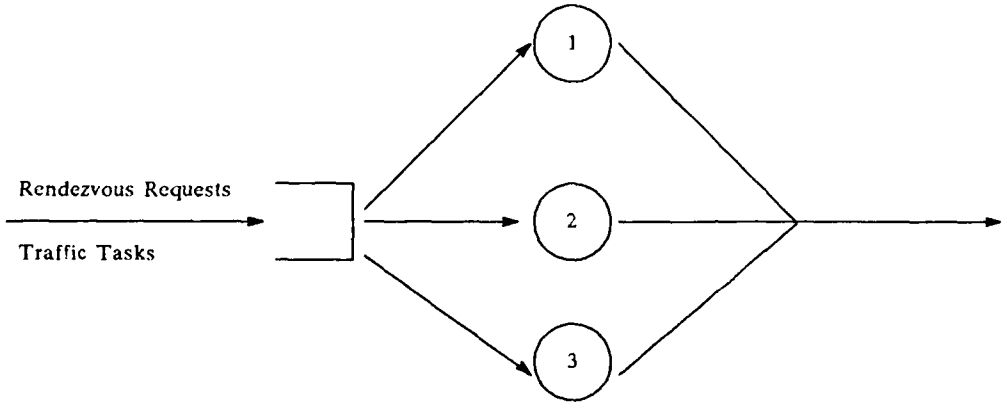

Figure 15. Server Processor separated into three parallel facilities.

Inflate the service demands,

$$D'_{R(Type\ 1)} = D_{R(Type\ 1)}/(1 - \lambda_H D_H)$$

$$D'_{R(Type\ 2)} = D_{R(Type\ 2)}/(1 - \lambda_H D_H)$$

and use these new values along with D_H to define the service demands at Facilities 1, 2, and 3, respectively, for the customers that they must service.

Solve for the residence times of the two rendezvous requests classes. This will be used to predict Rendezvous Response Times for the client tasks (Type 1) and client tasks (Type 2).

SERVER_TASK WITH TWO ENTRIES IN SERIES

Suppose SERVER_TASK has two points of entry in series. SERVER_TASK has the form

```
task body SERVER_TASK is
begin
    loop

      accept Entry_One (. . .) do
      . . .
      end Entry_One;

      accept Entry_Two (. . .) do
      . . .
      end Entry_Two;

    end loop;

end SERVER_TASK.
```

SERVER_TASK will hold two entry queues; one for Entry_Onc, and the other for Entry_Two. SERVER_TASK will execute a rendezvous with requests from each queue in an alternating manner. Each time SERVER_TASK gains control of Server Processor, it will first complete one rendezvous with a request from the Entry_One entry queue, and then complete one rendezvous with a request from the Entry_Two entry queue. If there are no requests at the Entry_Two entry queue, SERVER_TASK will release Server Processor and requeue for access when a request does enter the Entry_Two entry queue. At Client Processor, the client tasks (Type 1) and the client tasks (Type 2) will make their rendezvous requests to the Entry_One and Entry_Two entries, respectively.

Define the following notation:

1. $\lambda_{R(Type\ 1)}$ = Arrival Rate for Client Tasks (Type 1) to Client Processor.
2. $\lambda_{R(Type\ 2)}$ = Arrival Rate for Client Tasks (Type 2) to Client Processor.
3. λ_H = Arrival Rate for Traffic Tasks to Server Processor.
4. $D_{R(Type\ 1)}$ = Service Demand for SERVER_TASK to Complete One Rendezvous Request from the Entry_One Entry Queue.
5. $D_{R(Type\ 2)}$ = Service Demand for SERVER_TASK to Complete One Rendezvous Request from the Entry-Two Entry Queue.
6. D_H = Service Demand for Traffic Tasks of Server Processor.

We assume that $\lambda_{R(Type\ 2)} \leq \lambda_{R(Type\ 2)}$. Server Processor is separated into three parallel service facilities as in figure 15. Facility 1 will process the rendezvous requests of Client Tasks (Type 1), Facility 2 will process the rendezvous requests of Client Tasks (Type 2), and Facility 3 will process the traffic tasks.

Inflate the service demands

$$D'_{R(Type\ 1)} = D_{R(Type\ 1)}/(1 - \lambda_H D_H)$$

$$D'_{R(Type\ 2)} = D_{R(Type\ 2)}/(1 - \lambda_H D_H)\ ,$$

but in addition, we inflate again

$$D'_{R(Type\ 1)} = D'_{R(Type\ 1)}/(1 - [\lambda_{R(Type\ 1)} - \lambda_{R(Type\ 2)}]D_{R(Type\ 2)})$$

to take into account that requests queued at the Entry_One entry must wait an additional amount of time since SERVER_TASK is forced to service an equal number of requests at both entries. This value is used along with $D'_{R(Type\ 2)}$ and D_H to define the service demands for the customers at Facilities 1, 2, and 3, respectively.

Solve for the residence times of the two rendezvous requests classes. This will be used to predict Rendezvous Response Times for the client tasks (Type 1) and the client tasks (Type 2).

SERVER_TASK WITH A RENDEZVOUS WITHIN A RENDEZVOUS

Suppose SERVER_TASK rendezvous with a task while in rendezvous with another task, i.e., it has the form

```
task body SERVER_TASK is
begin
    loop

      accept Entry_One (. . .) do
      . . .

      accept Entry_Two (. . .) do
      . . .
      end Entry_Two;

      . . .
      end Entry_One;

    end loop;
end SERVER_TASK;
```

This case breaks down into Example B since execution of the code between "end Entry_Two;" and "end Entry_One;" will be done immediately after the Entry_Two rendezvous is completed.

CONCLUSION

Two sets of algorithms are proposed that solve open and closed rendezvous networks. For the open model, the solution is simple and requires only four input parameters. Therefore, quick, direct solutions are allowed. The closed-model solution is more involved but the techniques are still simple. Future work will extend these techniques to more complex rendezvous situations.

REFERENCES

Agrawal, S. C., and J. P. Buzen. May 1983. "The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems," *ACM TOCS* 1, 2, pp. 116-143.

Jacobson, P. A., and E. D. Lazowska. Feburary 1982. "Analyzing Queueing Networks with Simultaneous Resource Possession," *Comm. ACM*, vol. 25, no. 2, pp. 142-151.

Kleinrock, L. 1975. *Queueing Systems, Volume I : Theory*, John Wiley & Sons, Inc., New York.

Lazowska, E. D., J. Zahorjan, G.S. Graham, and K.C. Sevcik. 1984. *Quantitative System Performance*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Reiser, M., and S.S. Lavenberg. April 1980. "Mean Value Analysis of Multichain Queueing Networks," *JACM*, vol. 27, no. 2, pp. 313-322.

United States Department of Defense. 1983. "Reference Manual for the Ada Programming Language," ANSI/MIL-STD-1815A. (Ada is a registered trademark of the U.S. Government.)

Woodside, C. M., E. Neron, E.D.-S. Ho, and B. Mondoux. 1986. *An 'Active Server' Model for the Performance of Parallel Programs Written Using Rendezvous*, Carleton University, Ottawa, Canada.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1991	Final

4. TITLE AND SUBTITLE

PERFORMANCE MODELING OF THE ADA RENDEZVOUS

5. FUNDING NUMBERS

PE: 0602936N
WU: DN300189

6. AUTHOR(S)

A. E. Sterrett, M. K. Minei

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152–5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NOSC TD 2194

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Chief of Naval Research
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report details the analytical development of techniques that will predict the performance of systems using the rendezvous feature.

14. SUBJECT TERMS

Rendezvous Response Time
client tasks
Mean Value Analysis
two-processor system
server processor

15. NUMBER OF PAGES

34

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
A. E. Sterrett.	(619) 553-4078	Code 411

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0142	K. Campbell	(1)
Code 0144	R. November	(1)
Code 41	A. Justice	(1)
Code 411	A. Sterrett	(5)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NOSC Liaison Office
Arlington, VA 22217-5000

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research & Development
Information Center (NARDIC)
Alexandria, VA 22333

Navy Acquisition, Research & Development
Information Center (NARDIC)
Pasadena, CA 91106-3955